TRANSFER COST OF VIRTUAL MACHINE LIVE MIGRATION IN CLOUD SYSTEMS

Leszek Sliwko and Vladimir Getov
Distributed and Intelligent Systems Research Group
University of Westminster, 115 New Cavendish St., London W1W 6UW, U.K.

Leszek.Sliwko@my.westminster.ac.uk, V.S.Getov@westminster.ac.uk

Technical Report, November 2017

Abstract: Virtualised frameworks typically form the foundations of Cloud systems, where Virtual Machine (VM) instances provide execution environments for a diverse range of applications and services. Modern VMs support Live Migration (LM) – a feature wherein a VM instance is transferred to an alternative node without stopping its execution. The focus of this research is to analyse and evaluate the LM transfer cost which we define as the total size of data to be transferred to another node for a particular migrated VM instance. Several different virtualisation approaches are categorised with a shortlist of candidate VMs for evaluation. The selection of VirtualBox as the best representative VM for our experiments and analysis is then discussed and justified. The paper highlights the major areas of the LM transfer process – CPU registers, memory, permanent storage, and network switching – and analyses their impact on the volume of information to be migrated which includes the VM instance with the required libraries, the application code and any data associated with it. Then, using several representative applications, we report experimental results for the transfer cost of LM for respective VirtualBox instances. We also introduce a novel Live Migration Data Transfer (LMDT) formula, which has been experimentally validated and confirms the exponential nature of the LMDT process. Our estimation model supports efficient design and development decisions in the process of analysing and building Cloud systems. The presented methodology is also applicable to the closely-related area of virtual containers which is part of our current and future work.

Keywords: Cloud computing, transfer cost, live migration, platform as a service, service brokerage

1. INTRODUCTION

Cloud systems are unique in their elasticity, that is, their ability to dynamically allocate available resources to applications is unprecedented in computer science history. This elasticity is backed up by large-scale virtualisation, where every aspect of an application runs in a dedicated Virtual Machine (VM) environment. Applications are executed in VM instances and it is possible to move the VM instances around and to place them easily within another node. VM instances can be migrated 'cold', whereby an instance is suspended, transferred to alternative node and resumed [26]. Modern VMs such as XenServer [5], VirtualBox, and VMware also support Virtual Machine Live Migration (VM-LM), where VM instances are migrated on the fly. In this case, there is no offline period, except for a short final synchronization pause of around 60 to 300ms [8].

Historically, the VM-LM technology was debuted by VMware with the introduction of vMotion and GSX Server in 2003. Soon, other vendors attempted to develop VM-LM features of their own, for example Microsoft added Quick Migration in its Windows Server 2008 Hyper-V (later renamed

to Live Migration) [27] and Citrix released XenMotion for XenServer in the same year. There have been several studies modelling various aspects of the transfer cost for VM-LM since then. The most notable examples of related work include:

- the impact of allocated VM memory size on the migration time [10][25][38];
- the impact of memory page dirtying rate on the migration time [24][34] and the downtime length [19][25];
- the effect of available network bandwidth on the migration time [4][37];
- the energy overhead required to perform VM-LM [6][19].

While these approaches are valid in general, they focus solely on the impact for a particular VM instance and consider only factors such as loss of performance or network packages, length of downtime or impact on users. However, the migration of VM instances also causes disruptions on the infrastructure level, especially when non-trivial volumes of data need to be transferred and clutter network bandwidth, which could be allocated to alternative processes. Therefore, the research work presented in this article focuses on VM-LM and evaluates the total volume of information to be migrated. The main contributions of this paper are the experiments and the Live Migration Data Transfer (LMDT) formula which helps to estimate the total size of data transferred over a network during the VM-LM process.

The rest of this paper is organised as follows – Section 2 categorises the virtualisation approaches, lists several widespread VMs that support VM-LM and justifies the selection of a representative VM which is used in the experiments. Section 3 presents an analysis of the five major areas of the LM transfer process namely: CPU registers, memory, permanent storage, network switching, and code structure and dynamics. Section 4 details the experiments with a variety of applications which were performed as part of this research. Section 5 explains the specifics of the LMDT formula, while the conclusions are covered in Section 6.

2. VIRTUAL MACHINES IN CLOUD SYSTEMS

A range of tools and virtualisation technologies for building Clouds exist at present [17][20]. The virtualisation solutions currently deployed across service providers are unfortunately not standardised and significantly differ in many aspects. Based on the initial classification [28], the existing virtualisation approaches can be divided into five categories as follows:

- Full virtualisation relies upon an on the fly in-kernel translation of privileged instructions to user-level instructions. This results in significant performance drop since binaries of applications and their libraries must be analysed and transformed during the execution.
- Paravirtualisation requires modification to the source code of the guest OS. All privileged instructions are replaced with function calls to the hypervisor services – 'hypercalls'. The biggest drawback of this method is the necessity to have access to the source code of the guest OS which is not always possible and may interfere with the intellectual property rights.
- Hybrid virtualisation – in this model the guest OS uses paravirtualisation for certain hardware drivers and full virtualisation for other features, for example the guest OS can take advantage of hardware support for nested page tables, reducing the number of hypercalls required for virtual memory operations. At the same time the guest OS can benefit from fast I/O operations via lightweight access to paravirtualised devices.
- Hardware-assisted virtualisation has the advantage of hardware-level support. Recent developments in microarchitecture such as Intel's VT-x and the AMD-V added several

processor-level and memory-level mechanisms which directly support virtualisation. This approach eliminates the need to hook and emulate privileged instructions by hypervisors, and the guest OS can run at its native privilege levels.

- Virtual containers (VC) is an OS level virtualisation methodology in which a specially patched kernel allows for multiple isolated user-spaces. Therefore, this solution could be considered as an advanced implementation of the chroot operation. Nevertheless, from users' point of view, it is perceived as a real server [12]. VCs impose almost none of virtualisation overhead costs, as they operate inside a single kernel. VCs are generally locked to a single kernel version such as Docker, LXC or OpenVZ [30][32].

For the purposes of this research work we have shortlisted six widespread VMs that support VM-LM. Our main selection criterion was to include only mature and optimised implementations of the VM-LM technology. While VM-LM was first introduced as far back as 2009, this feature is still available only as an experimental feature in many VMs. All selected VMs support a variety of platforms and generally have good compatibility with commonly available hardware (except for XenServer which requires certain hardware features to be available). The shortlisted VMs are as follows:

- XenServer [5] has become a very popular choice for Cloud systems and is currently being used as a primary VM hypervisor in several Cloud providers, including Amazon EC2, IBM SoftLayer, Liquid Web, GoGrid, Fujitsu Global Cloud Platform, Rackspace Cloud, and CloudEx [18].
- VirtualBox supports a wide set of host OS-es, namely Linux, Mac OS X, Windows XP and in its later versions, Solaris. In addition, there are ports to FreeBSD and Genode. Natively supported guest OS-es are almost all versions of Windows, Linux, BSD, OS/2, Solaris, and so on. To achieve the best possible integration, VirtualBox comes with a set of native drivers and system applications called ‘Guest Additions’.
- The WMware product is a line of hypervisors. Type 1 runs directly on hardware and Type 2 runs on OS such as Windows, Linux and Mac OS X. VMware supports VM-LM, but it does not emulate instruction sets for hardware components that are not physically present. Instead, it focuses on running CPU instructions directly on the machine. However, this feature might cause problems when a VM instance is migrated to a machine with a different hardware setup [21].
- A KVM (Kernel-based VM) component was merged into Linux mainline kernel version 2.6.20. For KVM to work, the CPU must offer hardware-assisted virtualisation support: Intel’s VT-x for the x86 architecture and VT-i for Itanium architecture or AMD-V for AMD processors. KVM currently supports saving/restoring the VM state and offline/online migrations. In addition, the VM-LM can be performed between AMD and Intel hosts.
- Hyper-V is also known as Windows Server Virtualisation services. It provides virtualisation services in Windows 8 or newer. Hyper-V is capable of running several unique instances called ‘partitions’, each with its own kernel. Although VM-LM is supported, it is quite restricted and has several limitations, namely that a VM instance can be migrated only between identical versions of Windows Server 2008. Further, only x64 or Itanium architectures are supported, and all cluster nodes must be on the same TCP/IP sub-net.
- Docker works on the principles of VCs and relies on the resource isolation features of the Linux kernel such as cgroups and kernel namespaces. Docker enables the creation of multiple independent VCs to run within a single Linux instance [22] with each of them seen as a full OS, capable of running services, handling logging, etc. At the time of writing, Docker did not support LM – the integration with the Checkpoint/Restore In Userspace

(CRIU) tool does not allow the migration of a running application to the alternative container on the fly. However, recent publications [36] describe early experiments with LM feature while a working prototype has also been demonstrated in 2016 [13].

Table 1 presents a comparison of the selected VMs based on their core characteristics. It should be noted, that the host OS and the guest OS lists are not exhaustive; other OS-es may work without modifications. XenServer and WMware are implemented as type-1 (bare-metal) hypervisors which can work directly on hardware without the need for a host OS. Further information including more details about design decisions and principles of operation are usually proprietary and therefore not publicly available.

Virtual Machine	Virtualisation approach	Guest OS performance	Live Migration technology	Host OS	Guest OS	License
XenServer	Paravirtualisation	Native	XenMotion, Storage XenMotion	Windows, OS X x86, Linux	Windows 2008/7/8/8.1/10, CentOS, Red Hat/ SUSE/Oracle/Scientific/Debian Linux, Ubuntu, CoreOS	Open Source, Commercial
VirtualBox	Full virtualisation, Paravirtualisation	Close to native, Native	Teleporting	Windows, OS X x86, Linux, Solaris, FreeBSD	Windows 98/NT/XP/2000/Vista/2008/ 7/8/8.1/10, DOS, OS/2, FreeBSD, Mac OS Server, Solaris, Linux, Syllable, Ubuntu	Open Source
WMware	Full virtualisation	Close to native	vMotion	Windows, OS X x86, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS X, OS/2, DOS, CoreOS, BeOS, Darwin, Ubuntu, SUSE/Oracle Linux	Commercial
KVM	Full virtualisation *(requires AMD-V or Intel-VT-x)*	Close to native	Live Block Migration	Linux, FreeBSD, illumos	Windows Vista/2008/7/8/8.1/2012, CentOS, Red Hat/SUSE/Oracle Linux, Solaris	Open Source
Hyper-V	Full virtualisation	Close to native	Live Migration *(formerly: Quick Migration)*	Windows 8/Server 2012 (R2), Microsoft Hyper-V Server	Windows Vista/7/8/8.1/10/2008/2012 /2016, Red Hat/Oracle/SUSE/Debian Linux, Cent/OS, FreeBSD	Commercial
Docker	Virtual Containers	Native	*Working prototype (but not open source yet)*	Windows Server 2016 *(worker node)*, CentOS, Red Hat/ SUSE/Oracle Linux, Ubuntu	*(same as Host OS)*	Open Source, Commercial

Table 1: VMs Comparison

We believe that this comparison is sufficient to justify the selection of VirtualBox as a representative VM in VM-LM research for the following reasons:

- VM-LM is supported since version 3.1, therefore it is a mature solution. Upon detailed inspection of VirtualBox's source code, it was established that this feature's design does not differ substantially from other VM implementations.
- VirtualBox does not require any dedicated hardware while Type 1 hypervisors run only on a limited set of hardware.
- In comparison to other listed VMs, VirtualBox instances generally have fewer problems being migrated to nodes with a different host OS and different processor architecture. As an example of the opposite, Hyper-V and WMware require the same type of CPU architecture between source and target hosts.
- During VM-LM, VirtualBox transfers only the currently allocated and used by the VM memory, i.e. not the total memory configured for the VM. The difference in transferred data might be substantial if applications within the VM instance do not fully utilise the available VM memory [16].

- VirtualBox is widely used – there exist a number of freely available ports for all major operating systems such as Windows, Linux, Mac OS X, Solaris and FreeBSD. There are many publicly available guides and resolutions to issues available on the Internet. There are also many free out-of-the-box and pre-installed VirtualBox images available.
- One of the biggest strengths of VirtualBox is its ability to virtualise nearly any type of platform, including Linux, Windows, OS X, Solaris, Android, Chromium OS, etc. Furthermore, VirtualBox has universal good compatibility with hardware thanks to Guest Additions.
- Since version 4, VirtualBox has been released as an Open Source project and many fixes, improved stability and optimised performance patches have been contributed to its source code by active community.
- VirtualBox is used as a foundation for VC systems such as Docker; docker-machine tool provisions a specialised VirtualBox VM instance to run its kernel on Mac OS X.

3. LIVE MIGRATION

Both cold and live VM migrations are techniques for moving one VM instance from one node to another, usually of the same type. Depending on the vendor, various restrictions might apply, although in general the same CPU architecture is necessary and the VM will also perform a check of available features and extensions. This check together with the selection of the destination node can be supported much better if a node grouping approach is applied with a master controller node [9].

Cold migration requires stopping a VM and saving its state to snapshot files, moving these files to the destination and restoring the VM instance from a previously saved state. An obvious disadvantage of this approach is the unavoidable downtime required to stop the VM, transfer the state files and start the VM on the target node, when the service is not accepting requests.

However, modern VMs support a more advanced on the fly migration. The VM-LM feature – called 'teleporting' in VirtualBox [1] or 'vMotion' in VMware [21] – is a powerful functionality of modern VMs. The VM-LM dramatically reduces the downtime needed for virtualised applications and is the most suitable approach to achieving high-availability services. In essence, a continuously running VM instance is transferred to another VM running on a different physical machine without stopping its execution. This is done by transferring all VM memory and CPU registers on the fly, switching network devices to a new network address, and then either transferring the whole local disk storage content or reopening interfaces used in the network-attached storage (NAS). Therefore, the transfer cost of VM-LM depends on the specific application workload for the major areas of the migration process – CPU registers, memory, permanent storage, and network switching. Below, we discuss the key VM-LM challenges, as well as a performance analysis of their impact on the total migration cost for those four major categories of application workload.

3.1. COMPUTATION-INTENSIVE APPLICATIONS

Previous research [35] has shown that the amount of available CPU cycles on a machine has a substantial impact on the migration time. One of the tests demonstrates that assigning more CPU cycles to the VM-LM process often results in a reduction in the total migration time, but only to a point of around 50% CPU utilization. In our research, assigning more than 50% of CPU utilization did not shorten the migration time any further. Furthermore, our experiments have also shown that changes between 40% and 80% in the CPU utilization for different applications did not

noticeably affect the migration time. This can be explained by the relatively small size of the CPU registers and the CPU caches that need to be copied over.

3.2. MEMORY-INTENSIVE APPLICATIONS

Memory usage intensity has a huge impact on migration time. Memory migration can be achieved by directly copying memory to a target node. This process consists of copying all memory pages one by one onto a target node. If the content of a memory page is altered during migration, this memory page is marked as dirty which will result in another attempt to copy it again in the future. Generally speaking, during every migration, most of the pages which are not modified frequently will be copied once or a small number of times. However, a subset of memory pages commonly referred to as a Writable Working Set (WWS), will be altered in a very rapid manner – much faster than the speed at which network adapters can exchange information. These pages are often used for the stack and the local variables of the running processes as well as for network and disk buffers.

The usual solution in this scenario is for the VM-LM mechanisms to detect and mark hot pages by counting how many times a memory page has been dirtied during migration [17], and then to synchronise those remaining memory pages at the last stage while both VMs are paused. In the first migration round all memory pages are copied to the target VM while the source VM continues to run. During that round, some of the pages will have their content modified and will be marked as dirty. In the next round there will be a further attempt to copy them.

However, some memory pages are dirtied so rapidly so that they cannot be transferred over the network fast enough. Therefore, the final round pauses the source VM and all remaining pages are copied onto the target VM which subsequently starts while the source VM stops and shuts down. For an analysis and estimation of how big WWS is, see [8]. Also, specific research [35] has substantively examined what kinds of memory operations have an impact on WWS size. One of their testing scenarios has been split into two variants, that is, memory-read-intensive applications and memory-write-intensive applications.

The memory size initially allocated to an application had a linear impact on the migration time which is expected since more data had to be copied and there was no additional impact on the migration time when the memory-read-intensity increased. However, increases in memory-write-intensity did significantly slow down the migration process, albeit not linearly. When enough CPU cycles had been assigned to benchmark and the memory page dirtying ratio was high enough, the XenServer momentarily entered its final synchronization, and the migration time was not increased any further. It is difficult to provide actual numbers as to where the memory writing rate reaches a critical point. The existing research results [19] show that the memory writing rate started to significantly impact upon the migration time at around 800 Mbit/s, although these results are isolated to the specific research testing machine. I n practice, WWS is usually relatively small and VM downtime is barely noticeable for most applications. VM-LM usually requires a minimum stoppage before the next VM continues its execution at its final destination, providing a practically seamless migration of a running VM. For example, the XenServer requires only 60-300ms to perform final memory synchronization when migrating the Quake 3 server [9].

The same research shows, that during the VM-LM of the VM running SPECweb99 benchmark, only 18.2MB out of a total 676.8MB allocated memory was transferred in the final synchronization phase. The Quake 3 server needed to suspend the VM to transfer only 148KB of a total of 56.3MB allocated memory. Nevertheless, it is possible to design extreme scenarios where the VM is writing to a certain region of the memory faster than the network can transfer

it, for example the custom program 'MMuncher' resulted in the transfer of a huge 222.5MB out of a total of 255.4MB allocated memory in the final synchronization phase. There is a further significant difference depending on the implemented memory transfer method. More specifically, VMs such as KVM, VMware and VirtualBox transfer only the currently allocated and used by the VM memory, while other VMs such as Hyper-V and XenServer, transfer the total configured memory. This difference in transferred data might be substantial, even one order of magnitude (247-354MB vs. 2255-2266MB) of total transferred migration data [16].

From VM-LM's point of view, the most interesting module in VirtualBox is PGM (Page Manager and Monitor) which contains decision logic for triggering the VM suspension and for entering the final round of VM-LM. The decision is based on the estimated remaining dirty pages migration time, separately for the short-term average over the last four rounds and the long-term average based on all previous rounds. This algorithm then computes if the migration of all currently dirty pages exceeds the hardcoded 250ms maximum downtime. It should be noted, that the PGM module has not been updated since its initial implementation in 2010 and could be improved further, potentially reducing the total VM-LM time.

Researchers have also proposed certain optimization techniques which could reduce memory migration time. In [17], the authors propose the adaptive compression of migrated memory pages based on word similarity. Experiments show that most memory pages fall into either low or high word similarity, with the former usually a good candidate for fast compression algorithm. In [11], the authors suggest an ordered transfer of memory pages based on their dirtying rates as well as factoring in the available network bandwidth. The next migration iteration starts as soon as the accumulated dirtying rate exceeds the available bandwidth.

3.3. Disk I/O-intensive Applications

Storage migration might either involve transferring the whole storage over the network or only reopening the connection to a configured NAS device. Modern Cloud systems typically implement NAS as a preferable storage option, since it has many advantages such as centralised administration, a variety of standardised solutions across many different vendors and reduced failure rate features. If any virtual devices are mapped on local storage media, they also need to be migrated together with the VM instance. The VM is generally not aware of higher level data structures from the guest OS such as files or memory segments. The VM only reads and/or writes its serialised form as a stream of bytes. Therefore, the VM does not recognise which data is left over from previous operations but which is now marked as clean, meaning that the process saves everything. Consequently, upon migration, more data than indicated by the guest OS might need to be migrated since it involves copying the unused/dirty sectors.

Previous research has shown that it is possible to significantly exceed the available disk cache memory in order to force the VM to save data to actual persistent storage [35]. In this test, a sequential read pattern was used as it can generate more consistent I/O Operations Per Second (IOPS) compared to either sequential write or random read or write patterns. As with memory, an increase in IOPS caused an exponential increase in migration time, but the authors did not notice the existence of a 'break point', after which time further increases do not occur. This could be explained by a lack of monitoring of the disk sectors dirtying ratio implemented in VM. Generally speaking, memory operations have a few orders of magnitude wider bandwidth than respective I/O operations, especially for random access transfers.

3.4. Network-intensive Applications

For network resources, VM-LM relies on maintaining all open connections without the involvement of any kind of network traffic forwarding mechanism on the original node, since this may go offline and slow down responses from the host. VM-LM also copies the TCP stack state and, in addition, will carry IP addresses.

One suggested method [8] to resolve these challenges is to keep all network interfaces on a single switched LAN. This would allow the target node to generate an unsolicited ARP reply, broadcasting that the IP has moved to a new location and has a new Ethernet MAC Address. This would cause all routers to update their ARP cache and direct all network packages to the new host. An alternative solution is to migrate the Ethernet MAC Address together with other resources, relying on the network switch to detect the port change, although this may result in lost packages.

The impact of network utilization intensity on migration time has also been studied [19]. Migrating the VM's state between different physical nodes requires transferring a significant amount of data, and therefore the available network bandwidth is a valid concern. It can be noted that increases in network bandwidth utilization exponentially decreases migration time.

3.5. Code Structure and Dynamics

The discussion in Subsection III.B reveals that the number of VM-LM rounds and the amount of data transferred in each of those rounds is directly related to the size of WWS. Previous research [8][16][35] suggests that the memory write operations are the core cause of the repeated migrations of memory pages forming WWS. However, those investigations do not specify which applications are more prone to migrate harder and how the VM-LM is actually impacted.

The WWS is heavily impacted by the exact instruction composition of the application's compiled native code, or, as in the case of Java Virtual Machine (JVM), the interpreted byte code which is then compiled to the executable code. Frameworks equipped with automatic memory management (AMM) such as JVM's Garbage Collector (GC) [33] frequently move memory pages as new objects are created and are then released, which is a very common pattern within object-oriented programming. Furthermore, the frequency and the distribution of modified memory addresses strongly depend on the code's design and its style of programming. For example, the frequent use of immutable objects and singletons results in somewhat more static memory, while the common use of mutable objects and dynamically growing collections such as linked lists lead to bigger WWS, meaning that the application will be migrated harder. The reuse of existing objects from object pools ('flyweight' pattern) has the potential to lower the WWS size. Nevertheless, it is extremely challenging to predict what kind of behaviour VM-LM will demonstrate without knowledge of the technology stack and a detailed analysis of the application code.

While experimenting with the VM-LM process of various applications, a non-linear relation of the size of WWS to LMDT was noted. However, as shown below in the details of the experiments, this phenomenon is application-specific. For some programs it is barely noticeable, while for others it is clearly visible. As demonstrated in sections below, rapid exponential increases in LMDT are clearly visible for the busy backend application running in VMs which use AMM, such as JVM's GC. This is the result of massive memory movements during the memory reclamation phases of GC and, therefore, higher amount of dirtied memory pages which need to be copied in the next VM-LM round.

4. EXPERIMENTS

This research focuses on finding which parameters would significantly impact the size of the LMDT. Therefore, based on the above analysis, several scenarios were used for our experiments as presented below.

4.1. CONFIGURATION

The scope of these experiments has been necessarily limited by the hardware which is accessible in the laboratory and the available network infrastructure. The tests were designed with a focus on Open Source solutions which gave the option to examine the source code and better understand the inner workings of migrated applications.

Experiments were performed on a 100BASE-T network consisting of two low-end machines Amilo Pro V3515 with 2GB 533MHz memory, a router and a Network Attached Storage (NAS) device exposing a shared folder via Samba server via the Common Internet File System (CIFS). As such, the testing network was fully isolated and the network noise was minimised. The testing machines had Kubuntu 15.04 installed as the host OS, using the default configuration. The experiments used VectorLinux 6.0 Standard 'Voyager' as the guest OS, this being a lightweight and slimmed down Linux distribution designed to avoid resources overhead on existing system services and daemons.

In order to perform the experiments, it was necessary to define VM instances in VirtualBox. Each instance consists of two objects, the VM instance configuration and the virtual disk image (VDI) file which is placed in a remote shared folder. VirtualBox requires the identical definition of a particular VM so as to exist on both source and target hosts.

Data transfer measurements were taken with the help of the iptraf tool. Used bandwidth was measured separately for sent and received data and then totalled. The measurement error was only 1.96%; when exactly 100MB of random traffic was sent, the iptraf recorded transferred data (n=20) averaged 84790KB (s=1654KB).

4.2. EXPERIMENTAL SCENARIOS

Cloud systems allow users to deploy a very wide range of applications and services. In order to have a wide variety of applications this experiment was performed with the following configurations:

- An idle VM with only basic kernel canonical services running and a simple Xfce-4.4.3 Window Manager. During VM-LM, a whole environment is migrated – the Operating System and running service itself. Therefore, this configuration was used as a reference point and it was possible to measure the impact of only guest OS on migration.
- SPECjvm2008 is a benchmark program suite for measuring the client side Java runtime environments which was released by the Standard Performance Evaluation Corporation in 2008. It consists of 38 benchmark programs focusing on core java functionality and grouped into eleven categories and has wide a variety of workloads, from computation-intensive kernels to XML file processors [23]. The SPECjvm2008 workload mimics a variety of common general-purpose application computations. For a more detailed study of SPECjvm2008 performance see [29]. In the experiment, SPECjvm2008 benchmark ran on Java 1.6.0_06-b20.
- The Apache HTTP Server (Apache) is the most widely deployed Internet web server. As of November 2017, Apache is running 44.55% of all active websites [3]. Apache is often used with a range of technologies such as PHP, Perl, Python and frameworks such as

WordPress. Apache is available as open-source software released under the Apache License, for a wide number of OS-es. In this experiment, static content was deployed and an external machine with a JMeter (v2.13) used to simulate user traffic.

- In a typical online system, the data are stored in a persistent storage component, usually a database. This experiment examined the impact of the VM-LM process performed while PostgreSQL version 9.2.24 database was running 'select' and 'update' queries on a large database table. PostgreSQL is a popular database, with a market share of 26.5% of all actively used databases worldwide [2].
- VM Allocator is a custom application used to simulate a running application with a static read only memory area and sizeable WWS memory. Such an approach enabled the configuration of an exact set of dirtying pages and their ratio to total memory; therefore, experiments could be conducted with higher confidence. To make simulation more realistic, the VM Allocator ran several threads in parallel.

4.3. IDLE VM

To analyse the impact of allocated/used memory in VM-LM, the first experiment was migration of the same guest OS system and running applications in three different VM configurations: 256 MB, 512 MB and 1024 MB. No other parameters such as the number of CPU cores, the enabled CPU features (PAE/NX, VT-s/AMD-V), the configured peripherals, have been altered.

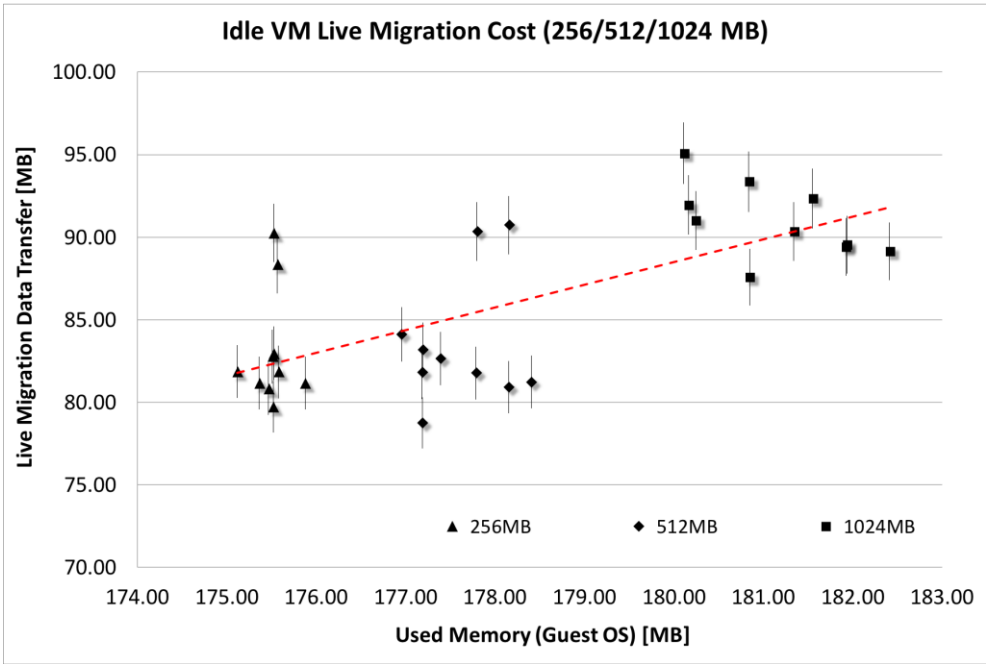

Figure 1: Idle VM Live Migration (256/512/1024MB)

Figure 1 presents VM-LM of an idle VM. This test helped to observe a slight increase in the allocated kernel memory while there was also an increase of the memory allocated to the VM. This is the effect of the Linux kernel requiring about 2 MB of memory for managing every additional 256 MB of memory. In this setup the memory stack size was set to 8192 bytes. It can also be noted that there was a minor increase in the total data transferred during the VM-LM. However, adding additional memory to idle the VM instance has only a minimal impact on the total transferred data: migrating an idle VM with 256 MB memory transferred about 80 MB and increasing the size of the configured VM memory from 256 MB to 1024 MB resulted in only around 10 MB more data being transferred.

4.4. APACHE HTTP SERVER

In this scenario we deployed the Apache HTTP Server (v2.4.18 compiled with APR 1.5.2) in the guest OS. The Apache server was configured to serve static content (10 MB of images) over the http channel. To have a reference point, an idle Apache HTTP Server instance was measured initially.

Transferring a VM instance using an idle Apache HTTP Server instance required 175 MB of network traffic. To simulate increasing user traffic, multiple continuous requests were generated with JMeter deployed on the external machine. JMeter is software designed to load test functional behaviour and measure performance of Web Applications such as web servers or services. In this research, JMeter simulated from 50 to 250 concurrent users, ca. 65 to 220 requests per second. It should be noted that the requested content was static meaning that the additional allocated memory was mainly to support the increasing number of simultaneously open connections.

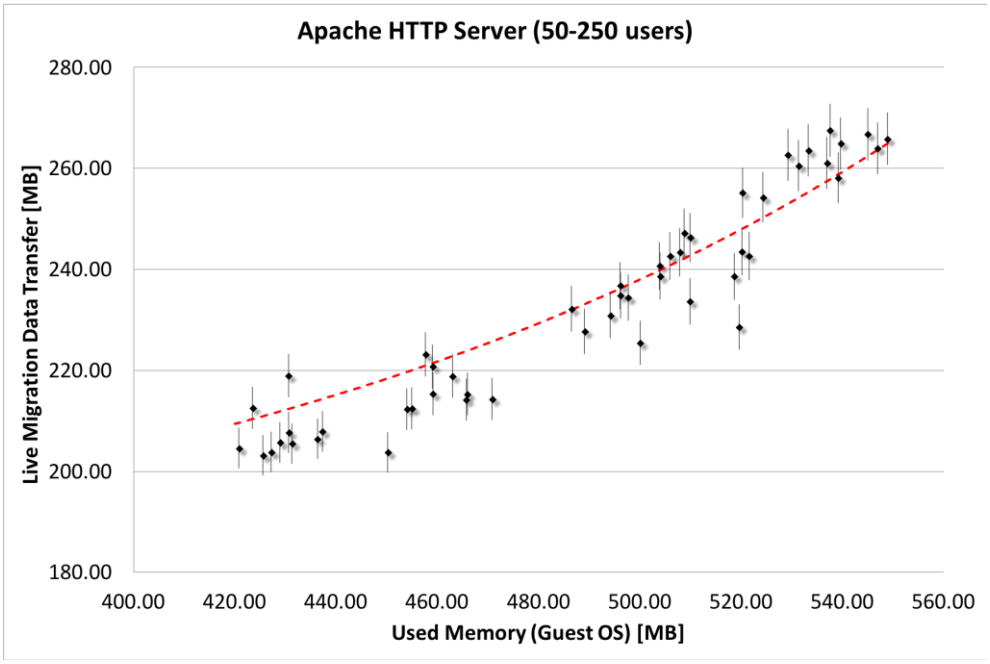

Figure 2: 50-250 Users Apache HTTP Server Live Migration

Figure 2 demonstrates the almost linear correlation between the total transferred data and memory utilization. It should be noted that opening and handling additional TCP connections is processed in the system's TCP/IP stack which could impact the size of the canonical memory allocated by the Linux kernel. This test scenario produces a near-linear correlation, as the migrated service is light on computations and the writable memory set size is rather constant. Therefore, an additional SPECjvm2008 experiment was used to examine how CPU-intensive applications behave during VM-LM.

4.5. SPECJVM2008

The experiments with CPU-intensive applications involved the SPECjvm2008 benchmark suite. SPECjvm2008 evaluates the performance of encryption and decryption implementations, various compression methods, floating point operations, objects serialization/deserialization, graphics visualization, XML transformations and others. Therefore, this suite performs a set of backend-heavy operations. Similar to the previous test with the Apache HTTP Server, it was necessary to firstly examine the impact of the VM memory size on data transfer. In order to force the loading

and caching system libraries, a single SPECjvm2008 process was run initially. It should be noted that SPECjvm2008 batch files were configured with a 96 MB maximum heap space. Java, by default, allocates a fourth of the available physical memory upon starting which might interfere with the running of as many as eight benchmark processes in parallel.

The test deployed an increasing number of SPECjvm2008 instances which were executed on a single VM machine. The main reason for this test was that those processes are separated; therefore, each of them will increase the WWS by a comparable size. Figure 3 demonstrates the test results which are visibly clustered in groups denouncing from 1 to 8 instances executed in parallel. Aside from the first SPECjvm2008 process which loads up about 32 MB of libraries, each additional SPECjvm2008 process allocates additional ca.15.5 MB of memory. The increase in transferred data is visibly exponential.

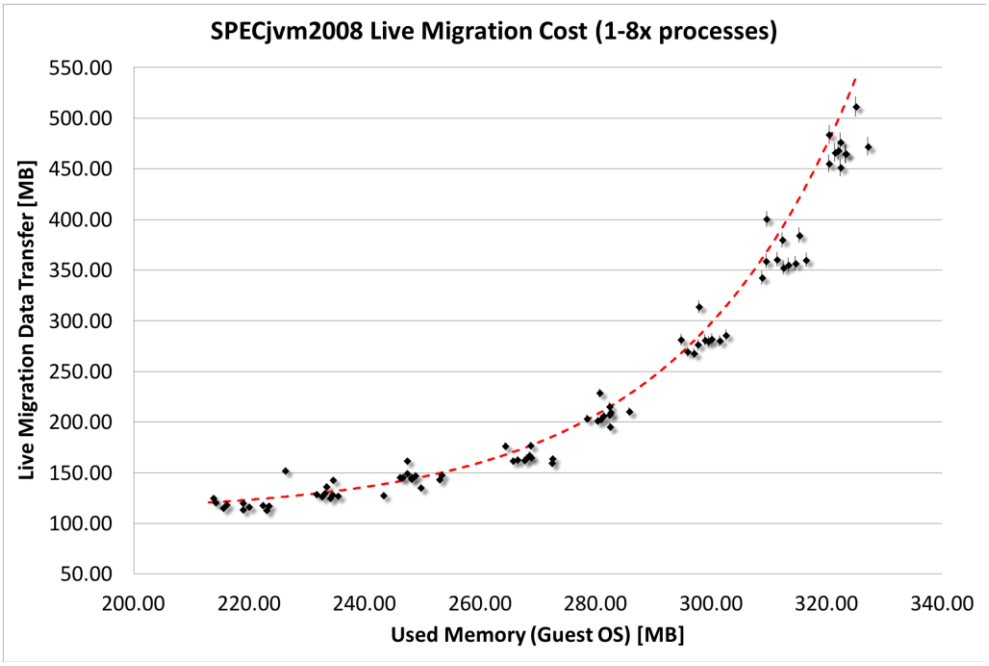

Figure 3: SPECjvm2008 Live Migration (1-8x processes)

This test also highlights a relative inefficiency when the active Java applications within a VM instance are being migrated. Good results are hard to achieve due to the increased memory movements caused by Java's GC. A solution to remedy such scenarios has been proposed [15] – a custom kernel module pauses JVM and invokes the GC just before each VM-LM round, then only objects from the tenured generation are being copied. In the final VM-LM round with the VM entirely paused, all objects from both the young and the tenured generation of the heap are being copied. Presented results show significant reduction of the total VM-LM time by over 90%, compared to the vanilla VM migration.

4.6. PostgreSQL Database

A typical online system consists of a frontend application, its backend services provider and a persistent storage component, usually in the form of a database. Having examined a popular frontend application (Apache HTTP Server) and a simulated backend application (SPECjvm2008), the persistent storage component now needs to be examined. While the current IT universe offers a wide range of specialised database solutions, such as graph-, NoSQL-, object-oriented-, and document-oriented databases which are suited to different data models, the most commonly

used are still relational databases, such as Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and so on.

In this research, a PostgreSQL database was selected due to its popularity, ease of use, reliability, stability, wide range of supported platforms, and design fit for a high-volume environment, while also being an Open-source project. At the time of writing, the PostgreSQL is often rated as third or fourth in the popularity index, with a market share of 26.5% of all actively used databases worldwide [2]. Given this, it is a representative choice for experimentation.

The following SQL commands were used to generate a test data table with one million rows of randomly generated strings and then to apply index on one of the columns:

```
SELECT
  generate_series(1,1000000) AS id,
  md5(random()::text) AS dataRaw,
  md5(random()::text) AS dataIndexed INTO TestData;
CREATE INDEX TestData_idx ON TestData(dataIndexed);
```

The test data came in two types: unindexed and indexed using default PostgreSQL B-tree. The SQL query optimiser can use database indexes when filtering and retrieving data from database tables, meaning a reduction in response time. B-tree indexes support general comparison operations such as >, <, =, etc., and also partially 'like' clauses. B-tree indexes can also be used to retrieve data in a sorted order. The test database which was generated allocated 2.11GB of disk space. It was stored remotely to the NAS device and mapped locally.

The next trials were designed to measure changes in the data transferred during VM-LM as PostgreSQL was running an SQL query. Those experiments were the most challenging to register consecutive results since the PostgreSQL database relies on multi-level caching to speed up its operations. Its cache buffers were cleaned between tests by restarting the server daemon service. OS was also forced to first synchronise and then drop disk caches within the same commands flow.

The subsequent experiments were split into two groups, presenting different scenarios:

- Where ‘select’ queries were run first on an indexed data and then on unindexed data, with the ‘like’ clause appended in both scenarios. In the first scenario, the query engine used a previously created index and loaded only data for matching rows. In the second scenario, the query engine was forced to iterate through each of the table’s rows to collect results. The main assumption for this group was that the additional memory activity from loading all the data would significantly increase the size of WWS and, as such, the first scenario would result in a smaller LMDT.
- Where an ‘update’ query modified parts of a test table. Five separate scenarios were designed, updating 20%, 40%, 60%, 80% and 100% of consecutive table rows respectively. Updating larger data sets involves building larger database transactions logs and requires more intensive memory operations which results in the expansion of the WWS. Furthermore, the ‘update’ operations require the modification of remote database files which are accessed over the network, and changes must be additionally committed via the CIFS protocol, the mechanism which is the additional source of memory activity.

During experiments using the ‘select’ query, the PostgreSQL processes allocated ca.229MB in addition to the memory allocated by guest OS. Predictably, queries involving the indexed data

were executed much faster than queries executed on unindexed data, taking three and eight minutes respectively. Interestingly, there was no noticeable LMDT difference when executing 'select' queries on indexed and unindexed data columns, meaning that the size of the WWS remained roughly the same. The explanation for this behaviour is the extensive reuse of the memory cache buffers by PostgreSQL. Until buffers are not dirtied by data modifications, they can be rapidly released and reused. The PostgreSQL exposes 'pg_buffercache' view, which is useful for examining the inner workings of the buffering. The first noticeable aspect during the scenarios with 'update' operations was the considerable slowdown during the VM-LM process. Normally, the update of one million rows would take two minutes, and five minutes while VM was being migrated.

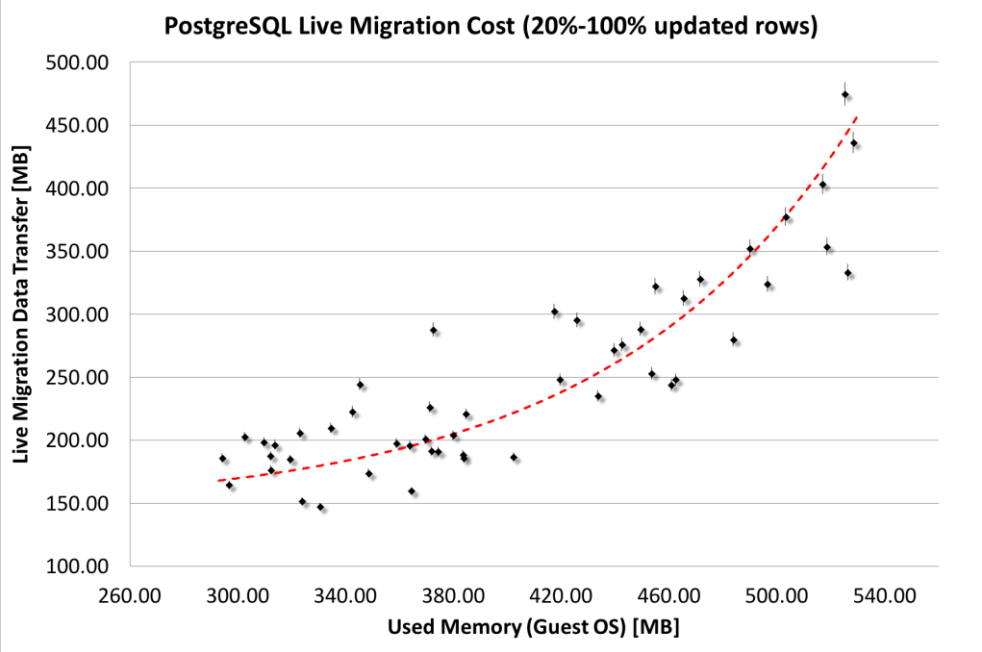

Figure 4: PostgreSQL Live Migration (20%-100% updated rows)

Figure 4 shows that processing 'update' operations, which altered 20% of rows, resulted in ca.310MB being allocated by PostgreSQL. Increasing the range of updated rows resulted in ca.40-55MB memory being further allocated by database processes for each additional 20% of all processed data rows. The allocated memory size changed very rapidly, and so only the range of memory changes is given. There was no noticeable difference between updating an indexed or unindexed data. This test emphasised the exponential nature of LMDT while migrating rapidly changing data. However, it also highlighted the difficulties of measuring the exact size of allocated memory and isolating WWS.

4.7. Custom VM-Allocator

The above scenarios test real-world scenarios, but it is difficult to measure the exact parameters of the running applications, such as the WWS size. Therefore, it was decided to implement a simple custom program, VM-Allocator, to help with the experiment. VM-Allocator allows specification of the Total Allocated Memory size (allocated and randomised upon the program's start) and WWS Size (part of the Total Allocated Memory and is continuously overwritten with random data). Several continuously working threads are used to write to WWS memory area.

In our experiments, there was a preference to avoid overwriting memory faster than the network could transfer it. This would result in very linear VM-LM data transferred increase. Since the WWS

memory area would be transferred every VM-LM round, the memory could only be finally synchronised in the final round when the VM is paused.

In this assessment (see Figure 5), the aim was to measure the impact of the WWS size on the transferred data. Therefore, it was necessary to test several different ratios of WWS vs. Total Allocated Memory – 10%, 20% and 30%. The increase in WWS size (without an increase of the memory overwriting speed) exponentially increases the LMDT size. The VM Allocator initialises all memory only once upon starting. Therefore, the measured used memory varies only marginally. As in previous examinations, the increase in transferred data is exponential.

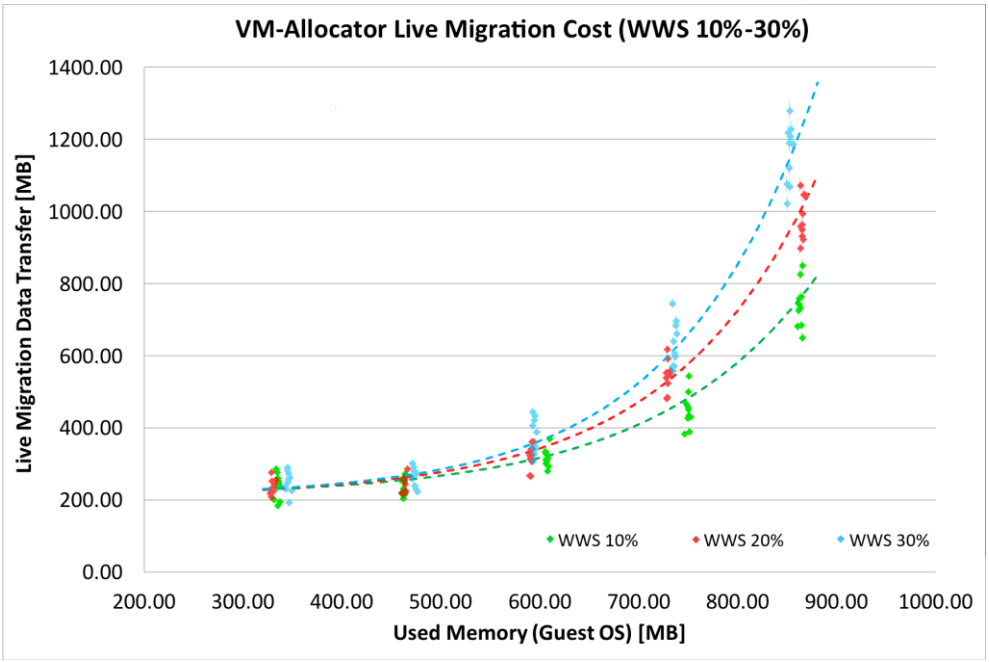

Figure 5: VM-Allocator Live Migration Cost (WWS 10%-30%)

Figure 5 presents our WWS test results. In this assessment, the aim was to measure the impact of the WWS size on the transferred data. Therefore, it was necessary to test several different ratios of WWS vs. Total Allocated Memory – 10%, 20% and 30%. The increase in WWS size (without an increase of the memory overwriting speed) exponentially increases the LMDT size. The VM Allocator initialises all memory only once upon starting. Therefore, the measured used memory varies only marginally. As in previous examinations, the increase in transferred data is exponential.

5. Live Migration Data Transfer Formula

Designing a method to accurately estimate live service migration time is not a trivial task. Nevertheless, a considerable amount of research has been done on the issue and several approximation models have been proposed [4][8][17][19] with very good results. In [29] a large real-time telecommunication application has been migrated in several scenarios and the results compared. In [37] the authors determined the network bandwidth required to guarantee a particular VM migration time.

It has been noted that the most feasible approach is to rely on historical data of memory dirtying rate for that particular service [19]. The memory access pattern of each application may vary depending on the actual utilization of the provided functionality. In such cases actual cost estimation may cause deviations if a model uses only previously sampled data. Experimental

results have proven that when an adaptive model relies on historical data yields, the results have a higher than 90% accuracy.

However, historical migration data is not always available due to new services, or services not yet migrated, or where the utilization of service has increased significantly, or where it has not been traced. In addition, changes in the environment may have a high impact on the migration time, for example the migration process itself consumes CPU cycles and network bandwidth, and a lack of these will slow down the migration process. Therefore, deriving a reliable VM-LM cost estimation formula is of utmost importance but.

Designing a general use formula for migration time is not feasible in practice. While total migration time is consistent for the same VM software, experiments show huge variances when migrating the same applications between different vendors. For example, the migration of the virtualised Ubuntu instance took between 7 and 23 seconds depending on the VM used [16]. This is also confirmed by further research [7][14][31].

One of the less researched factors in VM-LM is the actual size of data transferred. This has a direct impact on the Cloud infrastructure because every additional transferred byte limits the available bandwidth, and introduces noise to Cloud operations. Other factors such as decreased service performance, downtime, and increased CPU usage are local and isolated to a single VM instance or a particular physical machine at most. This experiment resulted in the following observations:

- The total configured VM memory has a small effect on the total transferred data. The reason for this is that the guest OS kernel allocates a small fraction of its memory to manage the total memory. In the test experiment, the kernel required about 2MB of memory to manage 256MB memory. Extending the VM memory from 256MB to 1024MB resulted only in the transference of only 10MB of additional data.
- The number of open network connections has minimal effect on the total data transferred. This is explained by the fact that current TCP/IP implementations are very mature, optimised and do not require many resources to perform network operations.
- Serving static content that does not require processing is economical in terms of VM-LM. Since data is mostly static, the majority of memory operations are read only. Therefore, transferring the memory page can be done only once, and thus the increase in transferred data during VM-LM is nearly linear.
- The high degree of computation activity by itself has no noticeable impact on the size of transferred data. However, the computation-intensive processes and services that significantly alter the memory state have the most substantial impact on the VM-LM data transfer size. When a memory page is repeatedly changed by write operations, the VM would repeatedly transfer it over the network. When the speed of dirtying those pages exceeds the available network bandwidth, those pages will be marked and must be transferred in the final round of migration.

From the above observations, the most significant factor in estimating the size of transferred data during VM-LM process is migrated application's allocated memory itself. Very rapid (i.e. faster than the network can transfer it) and fully overwriting WWS will result in this area of memory being fully migrated over and over again throughout all VM-LM rounds. Therefore, the increase of transferred data in this rare case is linear since WWS will always be transferred n-times. Based on our experiments, the following formula for the size of the LMDT has been devised:

$$LMDT = CMDT + MF * e^{(AF*AM)} \quad (1)$$

- **AM** (Application Memory):

$$AM = Total\ Used\ Memory - Canonical\ Memory \quad (2)$$

The Canonical (Kernel) Memory was measured together with any libraries loaded. System libraries and modules are loaded upon request and then are cached in the kernel memory and are shared by other applications.

- **AF** (Application Factor) – this parameter varies per application, and experiments show that the best approach is to estimate it by running a few simulations. As shown in (1), this parameter determines how steep the rise of LMDT is. In general, applications with complex logic modify memory pages more often and a significantly higher number of memory pages are being marked as dirty. This is especially true for AMM which tends to reallocate substantial amounts of data during the memory reclaim phase. The values presented in Table 2 were estimated from test experiments.

- **MF** (Migration Factor) – this parameter depends on infrastructure and it has been constant through all test experiments. In this experiment MF = 9.6 MB.

- **CMDT** (Canonical/Kernel Memory Data Transferred) – each VM and the contained applications transfer a certain chunk of data every time. This is measured when both the VM and the applications are idle as already described in Subsection IV.B. It should also be noted that the first instance of application increases the canonical memory since required libraries are loaded and cached in the memory. The estimated values from our test experiments are summarised in Table 2.

When applied to the experimental data, the above formula closely estimates the total size of the transferred data: for less computation-intensive applications, such as the Apache HTTP Server, the average approximation error is about ±4%. For computation-intensive codes, like SPECjvm2008, PostgreSQL and VM-Allocator, the average approximation error is between ±8% and ±11%.

Application (1024MB VM configuration)	CMDT	AF
Idle VM	90 MB	-
Apache HTTP Server	175 MB	0.00682
SPECjvm2008	115 MB	0.03305
PostgreSQL Database	145 MB	0.01072
VM-Allocator (WWS is 10% of total memory)	213 MB	0.00620
VM-Allocator (WWS is 20% of total memory)	213 MB	0.00676
VM-Allocator (WWS is 30% of total memory)	213 MB	0.00714

Table 2: Application Factors

6. CONCLUSIONS

This paper demonstrates that while the methodology for estimating the VM-LM cost should be tailored to a particular environment such as the deployed VM, network configuration, used storage types, available CPU cycles for migration, it is possible to find a reliable approach for the purposes of estimation. From previous research and our experimental results, several factors have been identified as having a significant impact on the service migration cost:

- The size of the allocated application memory and the size of WWS – memory-write-intensive applications are difficult to migrate since frequently modified memory pages must be migrated repeatedly over and over again. The size of WWS is often related to application design and its utilisation level.
- Frameworks equipped with AMM – this type of solutions are considerably harder to migrate. This is due to the significant amounts of data being re-allocated during the memory reclaim phase which results in larger numbers of memory pages being marked as dirty in each VM-LM round. As mentioned in Subsection IV.E there exist efficient strategies trying to address this weakness, however they require tight coupling with the application deployed within the VM which might not be always possible.
- The available spare CPU cycles on both the target and the source machines – migration is an OS process executed on the same physical machine as a hypervisor. The lack of available CPU cycles can create a bottleneck in the migration process itself [35].
- The size of the virtual drivers mapped to the local storage media and the IOPS rate – if data is stored locally and not on a NAS device, it needs to be transferred together with the VM state. Persistent data size can be substantial such as several TBs of data and can be a dominant factor in the migration time [16]. Frequent modifications in blocks might cause similar effect as in memory, where data is modified at a higher rate than it can be transferred. However, this effect is much less likely to happen.
- The network bandwidth available for migration – the migration process consists of transferring amounts of serialised data over the network to another physical node. The more network throughput can be allocated to migration, the faster it will complete. Additionally, a slow transfer rate might result in larger WWS, i.e. the memory dirtying rate exceeds network capabilities [8][19].

The presented VM-LM cost formula does not consider the cost of switching existing network connections to a new node. This is usually negligible since it is done in the hardware layer but depending on the implemented solution this additional cost might vary.

Nevertheless, it has been noted that given similar environments (i.e. VM vendor and version, comparable hardware specifications, same or similar application) there is little variety in the VM-LM impact, and historical data can be used to accurately estimate these metrics and parameters. The most practical way to obtain those values is through experimentation. Therefore, the working model should be capable of monitoring ongoing VM-LM and adjusting itself accordingly. Such a solution is presented in the literature [3][34] with good results.

Analysing the VM-LM cost is not simplistic and can have many dimensions and return various results based on the Cloud system. Therefore, it is necessary to apply appropriate limitations to the cost model and focus exclusively on the most important factors. As has been demonstrated in our experiments, estimating the LMDT size is not a trivial task, and many parameters need to be considered. Nevertheless, the results computed with our LMDT formula have an acceptable approximations level of up to ±19% in the worst-case scenario.

The scope of this research has been necessarily limited to the available software and hardware. All tested applications were executed on a single VM instance, while present systems tend to have more complex dependencies, such as distributed databases and file systems and 3rd party systems. Especially, the design of modern systems seems to follow the micro-services architecture principles. In such environments, loss of performance in one system component could easily propagate to other parts of the system.

REFERENCES

[1] "VirtualBox User Manual." Oracle Corporation. Available from: http://download.virtualbox.org/virtualbox/UserManual.pdf Retrieved October 30, 2017. Version 5.2.0.

[2] "Developer Survey Results 2017." Stack Overflow. May 13, 2017. Available from: https://news.netcraft.com/archives/2017/11/21/november-2017-web-server-survey.html Retrieved 11th November, 2017.

[3] "November 2017 Web Server Survey." Netcraft, Web Server Survey. November 21, 2017. Available from: https://news.netcraft.com/archives/2017/11/21/november-2017-web-server-survey.html Retrieved November 24, 2017.

[4] Akoush, Sherif, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper. "Predicting the performance of virtual machine migration." In Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on, pp. 37-46. IEEE, 2010.

[5] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. "Xen and the art of virtualization." In ACM SIGOPS operating systems review, vol. 37, no. 5, pp. 164-177. ACM, 2003.

[6] Callau-Zori, Mar, Lavinia Samoila, Anne-Cécile Orgerie, and Guillaume Pierre. "An experiment-driven energy consumption model for virtual machine management systems." Sustainable Computing: Informatics and Systems (2017).

[7] Chierici, Andrea, and Riccardo Veraldi. "A quantitative comparison between Xen and KVM." In Journal of Physics: Conference Series, vol. 219, no. 4, p. 042005. IOP Publishing, 2010.

[8] Clark, Christopher, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. "Live migration of virtual machines." In Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2, pp. 273-286. USENIX Association, 2005.

[9] Corsava, Sophia, and Vladimir Getov. "Self-healing Intelligent Infrastructure for Computational Clusters." In Proceedings of the SHAMAN Workshop, ACM ISC Conference, pp.1-8, ACM, 2002.

[10] Dargie, Waltenegus. "Estimation of the cost of vm migration." In Computer Communication and Networks (ICCCN), 2014 23rd International Conference on, pp. 1-8. IEEE, 2014.

[11] Du, Yuyang, Hongliang Yu, Guangyu Shi, Jian Chen, and Weimin Zheng. "Microwiper: efficient memory propagation in live migration of virtual machines." In Parallel Processing (ICPP), 2010 39th International Conference on, pp. 141-149. IEEE, 2010.

[12] Dua, Rajdeep, A. Reddy Raja, and Dharmesh Kakadia. "Virtualization vs Containerization to Support PaaS." In Cloud Engineering (IC2E), 2014 IEEE International Conference on, pp. 610-614. IEEE, 2014.

[13] Estes, Phil, and Shaun Murakami. "Live Container Migration on OpenStack." OpenStack Summit Barcelona 2016. Oct 25, 2016.

[14] Feng, Xiujie, Jianxiong Tang, Xuan Luo, and Yaohui Jin. "A performance study of live VM migration technologies: VMotion vs XenMotion." In Asia Communications and Photonics Conference and Exhibition, p. 83101B. Optical Society of America, 2011.

[15] Hou, Kai-Yuan, Kang G. Shin, and Jan-Lung Sung. "Application-assisted live migration of virtual machines with Java applications." In Proceedings of the Tenth European Conference on Computer Systems, p. 15. ACM, 2015.

[16] Hu, Wenjin, Andrew Hicks, Long Zhang, Eli M. Dow, Vinay Soni, Hao Jiang, Ronny Bull, and Jeanna N. Matthews. "A quantitative study of virtual machine live migration." In Proceedings of the 2013 ACM cloud and autonomic computing conference, p. 11. ACM, 2013.

[17] Jin, Hai, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. "Live virtual machine migration with adaptive, memory compression." In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, pp. 1-10. IEEE, 2009.

[18] Jin, Hai, Shadi Ibrahim, Tim Bell, Li Qi, Haijun Cao, Song Wu, and Xuanhua Shi. "Tools and technologies for building clouds." In Cloud Computing, pp. 3-20. Springer London, 2010.

[19] Liu, Haikun, Hai Jin, Cheng-Zhong Xu, and Xiaofei Liao. "Performance and energy modeling for live migration of virtual machines." Cluster computing 16, no. 2 (2013): 249-264.

[20] Luo, Jun-Zhou, Jia-Hui Jin, Ai-Bo Song, and Fang Dong. "Cloud computing: architecture and key technologies." Journal of China Institute of Communications 32, no. 7 (2011): 3-21.

[21] Marshall, Nick. Mastering VMware VSphere 6. John Wiley & Sons, 2015.

[22] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014, no. 239 (2014): 2.

[23] Oi, Hitoshi. "A preliminary workload analysis of specjvm2008." In Computer Engineering and Technology, 2009. ICCET'09. International Conference on, vol. 2, pp. 13-19. IEEE, 2009.

[24] Rybina, Kateryna, Waltenegus Dargie, Subramanya Umashankar, and Alexander Schill. "Modelling the live migration time of virtual machines." In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", pp. 575-593. Springer, Cham, 2015.

[25] Salfner, Felix, Peter Tröger, and Andreas Polze. "Downtime analysis of virtual machine live migration." Fourth Int. Conference on Dependability (DEPEND 2011). IARIA, pp. 100-105. 2011.

[26] Sapuntzakis, Constantine P., Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. "Optimizing the migration of virtual computers." ACM SIGOPS Operating Systems Review 36, no. SI (2002): 377-390.

[27] Savill, John. "Mastering Windows Server 2016 Hyper-V." (2016).

[28] Shirinbab, Sogand, Lars Lundberg, and Dragos Ilie. "Performance comparison of KVM, VMware and XenServer using a large telecommunication application." In Cloud Computing. IARIA XPS Press, 2014.

[29] Shiv, Kumar, Kingsum Chow, Yanping Wang, and Dmitry Petrochenko. "SPECjvm2008 performance characterization." Computer Performance Evaluation and Benchmarking (2009): 17-35.

[30] Smith, Randall. Docker Orchestration. Packt Publishing Ltd, 2017.

[31] Tafa, Igli, Elinda Kajo, Ariana Bejleri, Olimpjon Shurdi, and Aleksandër Xhuvani. “The Performance Between XEN-HVM, XEN-PV and OPEN-VZ During Live Migration.” International Journal of Advanced Computer Science and Applications (2011): 126-132.

[32] Tang, Xuehai, Zhang Zhang, Min Wang, Yifang Wang, Qingqing Feng, and Jizhong Han. "Performance evaluation of light-weighted virtualization for paas in clouds." In International Conference on Algorithms and Architectures for Parallel Processing, pp. 415-428. Springer, Cham, 2014.

[33] Urma, Raoul-Gabriel, Mario Fusco, and Alan Mycroft. "Java 8 in Action: Lambdas, Streams, and functional-style programming." Manning Publications, 2014.

[34] Verma, Akshat, Gautam Kumar, Ricardo Koller, and Aritra Sen. "Cosmig: Modeling the impact of reconfiguration in a cloud." In Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on, pp. 3-11. IEEE, 2011.

[35] Wu, Yangyang, and Ming Zhao. "Performance modelling of virtual machine live migration." In Cloud Computing (CLOUD), 2011 IEEE International Conference on, pp. 492-499. IEEE, 2011.

[36] Yu, Chenying, and Fei Huan. "Live migration of docker containers through logging and replay." In Advances in Computer Science Research, 3rd International Conference on Mechatronics and Industrial Informatics, pp. 623-626. Atlantis Press, 2015.

[37] Zhang, Jiao, Fengyuan Ren, Ran Shu, Tao Huang, and Yunjie Liu. "Guaranteeing delay of live virtual machine migration by determining and provisioning appropriate bandwidth." IEEE Transactions on Computers 65, no. 9 (2016): 2910-2917.

[38] Zhao, Ming, and Renato J. Figueiredo. "Experimental study of virtual machine migration in support of reservation of cluster resources." In Proceedings of the 2nd international workshop on Virtualisation technology in distributed computing, p. 5. ACM, 2007.